

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

# OPERATING SYSTEMS - CS235AI

### REPORT

**Submitted by**

**Shrinidhi I 1RV22CS193**

**Suraj Chanaveeragoudra 1RV22CS211**

Submitted To:

Dr. Jyoti Shetty

Assistant Proffessor

### Computer Science and Engineering 2023-2024

TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| SNO | CONTENT | PAGENO |
| 1 | Introduction | 3 |
| 2 | System Architecture | 5 |
| 3 | Methodology | 7 |
| 4 | System calls used | 9 |
| 5 | Output/result | 11 |
| 6 | Conclusion | 12 |

INTRODUCTION

Introducing the Not Recently Used (NRU) Page Replacement Algorithm

In the intricate dance of memory management within computer systems, the Not Recently Used (NRU) page replacement algorithm emerges as a graceful yet powerful performer. Picture a stage where memory is the spotlight, and pages are the dancers, each vying for a coveted spot in the limelight.

NRU, with its elegant simplicity, assigns pages to different classes based on their recent usage history. It deftly categorizes pages as either recently accessed or not, intuitively recognizing that pages left undisturbed are less likely to be called upon in the immediate future.

When the time comes to swap out a page, NRU plays the role of a discerning critic, selecting pages from the not recently used classes first, prioritizing those that have been referenced but not modified. This strategy aims to minimize disruptions, ensuring that the most relevant pages remain in memory, ready to take centre stage when needed.

Though NRU's performance is impressive, it is not without its nuances and challenges. Like any algorithm, its effectiveness depends on the context and the specific demands of the system. Yet, in the grand production of memory management, NRU's role is vital, contributing its unique flair to the intricate choreography of data movement in computer systems.

**PROBLEM STATEMENT AND OBJECTIVES**

Evaluate the performance and efficiency of the Not Recently Used (NRU) page replacement algorithm under varying memory configurations, comparing it with other established algorithms, to determine its advantages for practical implementation in modern operating systems.

**Objective**

Reducing Page Fault Rate: The primary objective of implementing NRU page replacement is to minimize the occurrence of page faults. By evicting pages that have not been accessed recently, the algorithm aims to keep frequently used pages in memory, reducing the need for disk I/O operations to retrieve pages from secondary storage.

Maintaining Fairness: NRU also aims to maintain fairness among different processes or users sharing the same memory space. By considering both the recent usage and the dirty/clean status of pages, it attempts to evict pages in a balanced manner across processes, preventing any single process from monopolizing memory resources.

Avoiding Thrashing: Thrashing occurs when the system spends a significant portion of its time swapping pages between main memory and disk due to excessive page faults. NRU helps in mitigating thrashing by efficiently managing page replacements based on recent usage patterns. By evicting less frequently accessed pages, it helps in maintaining a stable working set in memory, thereby avoiding thrashing scenarios.

System Architecture

Page Table: Each process in the system has a page table that maps virtual pages to physical frames. This page table includes additional bits for tracking the referenced and modified status of each page

Hardware Support: The CPU's memory management unit (MMU) or hardware page table walker is responsible for updating the referenced and modified bits whenever a page is accessed or modified. This hardware support ensures efficient tracking of page usage.

Page Replacement Algorithm: The operating system kernel includes logic to implement the NRU algorithm. This logic scans the page tables periodically or when a page fault occurs to select a victim page for replacement based on their class.

Page Replacement Queue: The operating system maintains a data structure, such as a queue or a list, to keep track of the pages eligible for replacement. Pages are organized into classes within this queue based on their referenced and modified status

Interrupt Handling: When the NRU algorithm selects a victim page for replacement, it may trigger a page fault interrupt. The operating system's interrupt handler is responsible for handling this interrupt and performing necessary actions, such as fetching the required page from secondary storage and updating the page tables.

Statistics Collection: Optionally, the system may collect statistics related to page accesses, modifications, and replacements for performance monitoring and tuning purposes.

PAGING:

|  |  |  |  |
| --- | --- | --- | --- |
| FNO | PNO | MB | RB |
| 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |
| 2 | 2 | 0 | 0 |
| 3 | 3 | 0 | 0 |

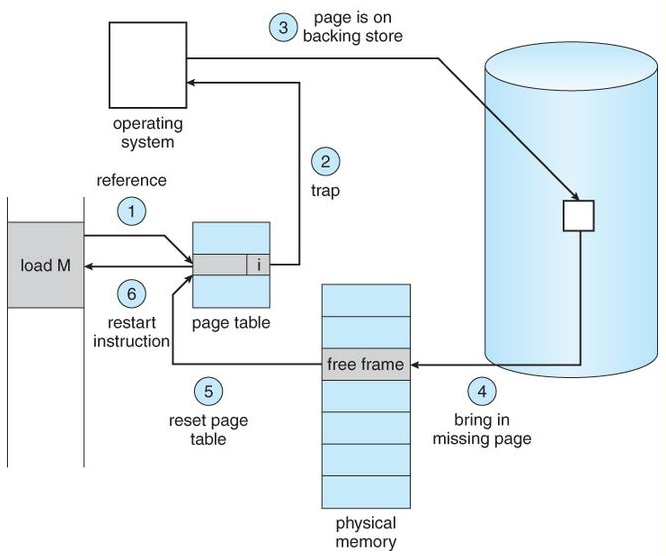
|  |
| --- |
| 0 |
| 1 |
| 2 |
| 3 |

|  |
| --- |
| Page 0 |
| Page 1 |
| Page 2 |
| Page 3 |

PAGES

PHYSICAL MEMORY

PAGE TABLE



*Fig Demand Paging Hardware*

Methodology

* A reference bit (R) and a modified bit (M) are given to every page in memory. Normally, each page table entry for a page contains these parts.
* The algorithm starts the NRU selection procedure at regular intervals or in response to specified events (such as clock ticks or timer interrupts).
* Based on their R and M bits, the algorithm sorts all of the pages into one of the following four classes-

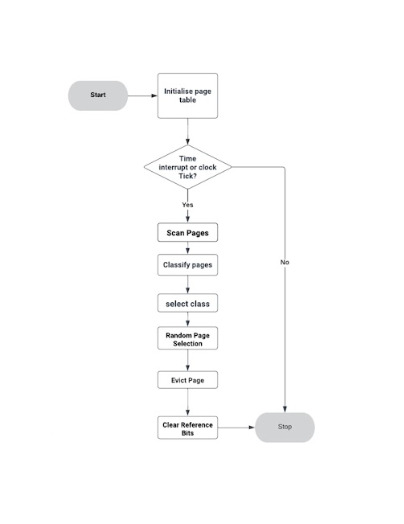
**Class 0**− Pages have R and M values of 0.

**Class 1**− Pages have R = 0 and M = 1.

**Class 2**− ages where R = 1 and M = 0.

**Class 3**− Pages where R and M both equal 1.

* A page is chosen for eviction by the algorithm after the pages have been classified. It starts by looking for the class with the lowest number that is not empty. The algorithm favors the class with the lesser number of pages when there are numerous classes. For instance, class 0 would be chosen if both class 0 and class 1 had pages.

The program picks a page at random for eviction from the designated class. This can be done by selecting a page at random after iterating over all of the pages in the class or by using a random number generator. A replacement page can then be loaded if necessary when the selected page is removed from memory. An appropriate modification is made to the related page table entry. Finally, to get ready for the subsequent selection cycle, all of the memory's remaining pages have their reference bits (R) cleared (set to 0). NRU tries to delete pages that are less likely to be needed in the near future by routinely scanning and evicting pages that have yet to be utilized (i.e., having a reference bit of 0). NRU does not take into account the frequency of page references, which can lead to a less-than-ideal performance in some circumstances.

System calls used

1. sem\_wait: sem\_wait decrements the semaphore pointed by sem. If the semaphore value is non-zero, the decrement happens right away. If the semaphore value is zero, the call blocks till the time semaphore becomes greater than zero and the decrement is done. sem\_wait returns zero on success and -1 on error. In case of error, the semaphore value is left unchanged and errno is set to the appropriate error number.

sem\_init(sem\_t \*sem, int pshared, unsigned int value);

sem\_wait(sem\_t \*sem);

1. sem\_post: sem\_post increments the semaphore. It provides the V operation for the semaphore. It returns 0 on success and -1 on error.

sem\_post(sem\_t \*sem);

1. sem\_open: sem\_open is the call to get started for a semaphore. sem\_open opens an existing semaphore or creates a new semaphore and opens it for further operations.

* The first parameter, name, is the name of the semaphore.The oflag can have O\_CREAT, in which case, the semaphore is created if it does not already exist.
* If both O\_CREAT and O\_EXCL are specified, the call gives an error, if the semaphore with the specified name already exists.
* If the oflags parameter has O\_CREAT set, the second form of sem\_open has to be used and two additional parameters, mode and value have to be specified. The mode parameter specifies the permissions for the semaphore, which are masked with the umask for the process, similar to the mode in the open system call for files.
* The last parameter, value is the initial value for the semaphore. If O\_CREAT is specified in oflag and the semaphore already exists, both the mode and value parameters are ignored.

#include <fcntl.h>

#include <sys/stat.h>

#include <semaphore.h>

sem\_t \*sem\_open (const char \*name, int oflag);

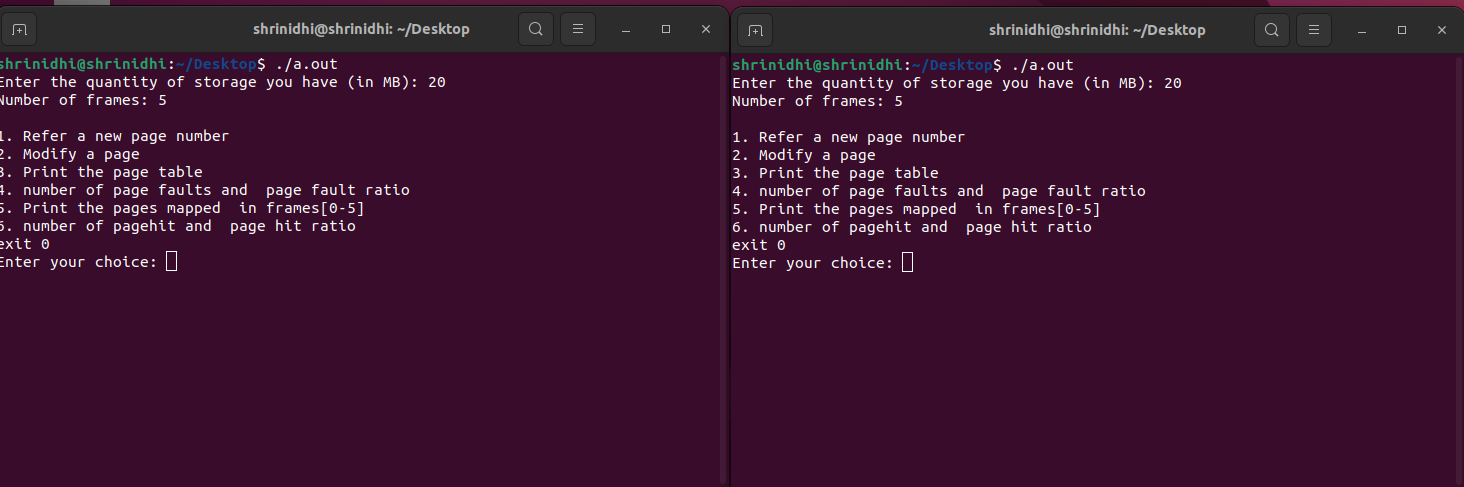
sem\_t \*sem\_open (const char \*name, int oflag, mode\_t mode, unsigned int value);

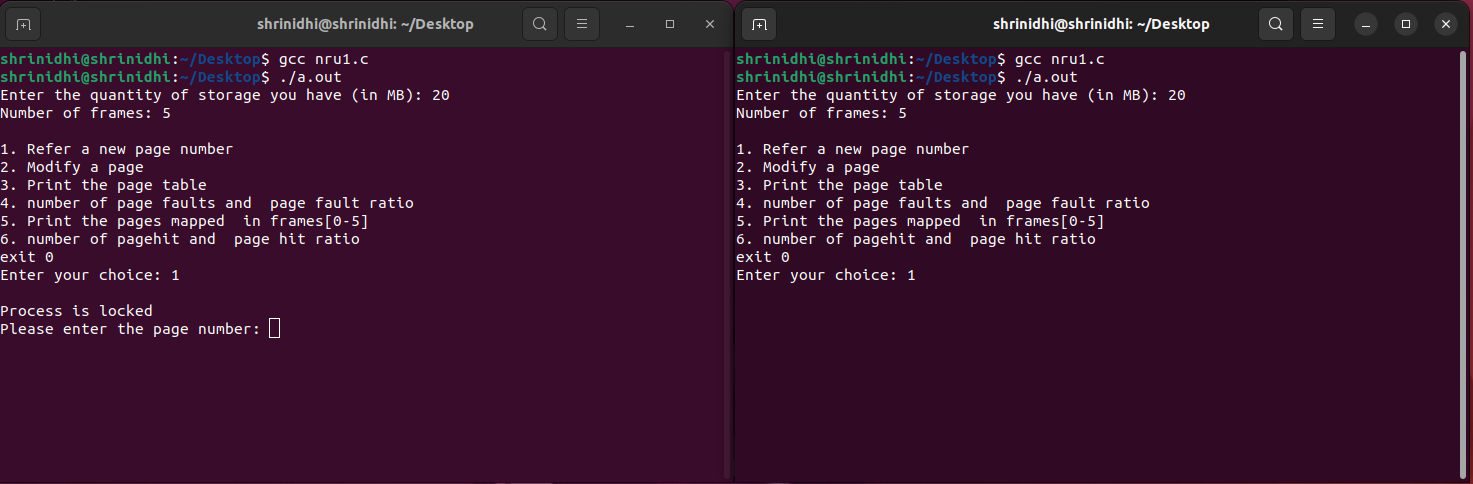
Execution steps:

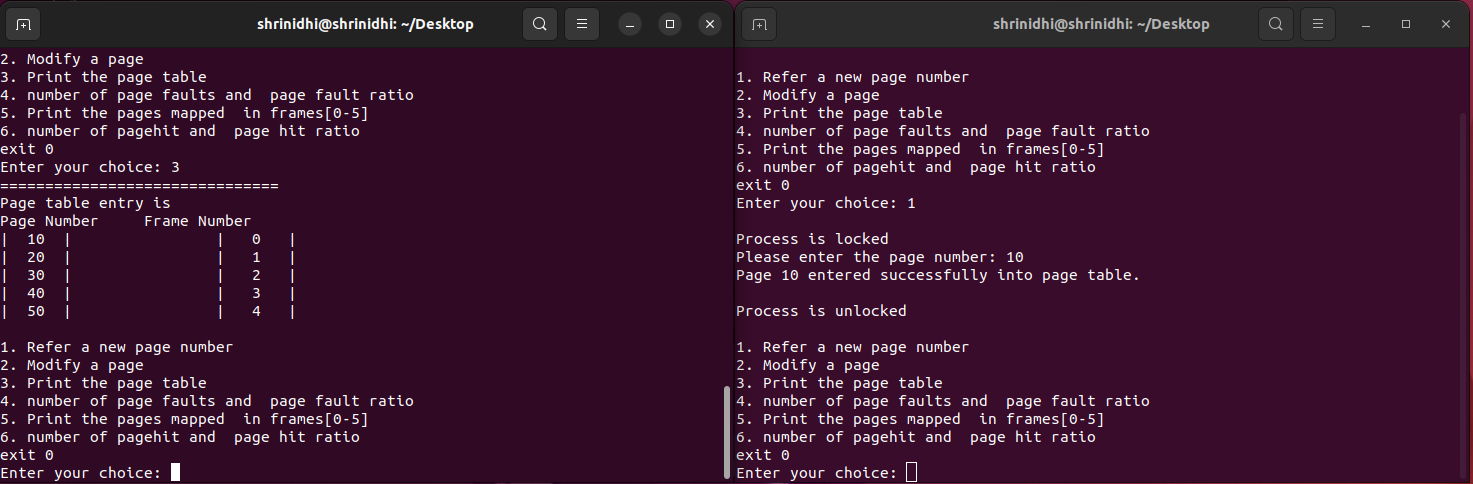
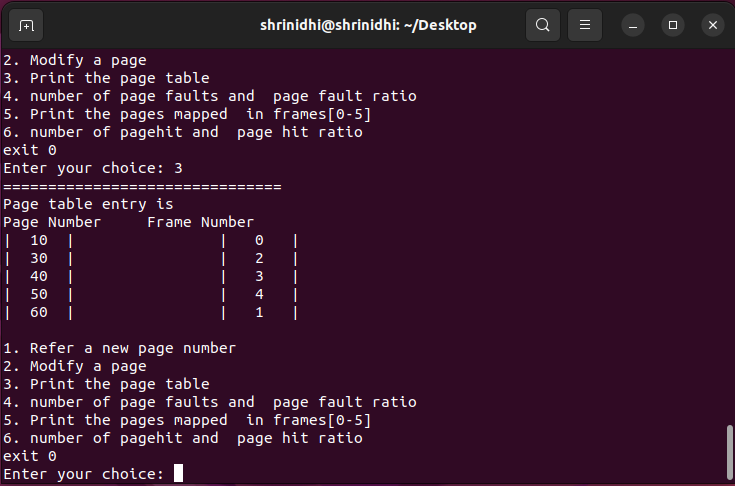
* Compile the program and generate the exe file <name>(nru)
* cc -nru.c -o nru
* ./nru

>>follow the menu driven instruction

Output/Result:





Here page 10 is modified. Hence page next to it page 20 is replaced by the new page 60.If

Page 10 is not modified/referenced the page 10 will be replaced.

Conclusion:

In conclusion, the Not Recently Used (NRU) page replacement algorithm stands as a foundational tool for optimizing memory management within operating systems. Its ability to categorize pages based on recent use history plays a pivotal role in reducing page faults and enhancing overall system performance. The algorithm's simplicity and low computational overhead make it particularly advantageous for systems with limited resources, allowing for efficient memory management without significant processing burden.

However, it's important to note NRU's limitations, such as the "two-time" problem, where infrequently accessed pages may not be promptly evicted despite recent use. To address this challenge, fine-tuning of class thresholds is essential to strike a balance between retaining frequently accessed pages and reclaiming memory space effectively. By carefully adjusting these thresholds, operating systems can optimize NRU's performance to better align with the specific access patterns of the applications running on the system.

In summary, while NRU may not offer the most advanced approach to page replacement, its effectiveness in minimizing fault rates and its compatibility with resource-constrained environments highlight its significance in enhancing operating system performance. By integrating NRU with other memory management techniques, operating systems can achieve efficient memory utilization and maintain optimal performance across diverse computing environments.